Scalar Multiplications

Implemented multiplications from Algorithms Illuminated Part 1 ( RoughGarden, Standford ) in Python

## Standard Multiplication

to perform multiplication as taught in school , we perform n² multiplication and n² - n additions so we perform at maximum 2n² operations ( assume you are doing a n \* n multiplication and n = 4 . then take the maximum numbers

### Standard Multiplication Algorithm

Coming soon

### Recursive Integer Multiplication Algorithm

input: two n-digit positive integers x and y

output: the product of x.y

Assumption: n is a power of 2 ( even), x and y have same number of digits

if n = 1 then

compute x\*y in one step and return the result

else

a,b := first and second half of x ,

c,d := first and second half y ,

recursively compute

ac := a\*c

ad := a\*d

bc := b\*c

bd := b\*d

compute 10^n \* ac + 10^(n/2) \* (ad+bc) )+ bd

## Karatsuba multiplication

Similar to the recursive multiplication algorithm above but instead of computing recursively both ad and bc and then taking their sum in the last step, we recursively compute the multiplication (a+b)(c+d), then we subtract ac and bd to get the sum of bc +ad to use it directly in the last step (Guass's trick)

### Kartsuba Multiplication Algorithm

input: two n-digit positive integers x and y

output: the product of x and y

Assumption: n is a power of 2 ( even), x and y have same number of digits

if n = 1, then computer will calculate x\*y in one step and return the result

else

a,b := first and second half of x ,

c,d = first and second half y,

recursively compute

ac := a\*c

a\_d\_b\_c := (a+b)(c+d)

bd := b\*d

ad\_plus\_bc = a\_d\_b\_c - ac - bd

compute 10^n \* ac + 10^(n/2) \* (ad\_plus\_bc )+ bd

## General Karatsuba Multiplication

The two algorithms above assume a 2^N number of digits. Both algorithms can be extended to handle an arbitrary number of digits ( odd and even ) . The idea is to pad the number of digits with zeros to the next 2^n number and then remove these zeros after the recursive muliplication happens. for example if you want to multiply two numbers with 9 digits each. First you get the next 2^ n , which is 16 in this case then you pad the 9 digit numbers with ( 16 - 9 ) zeros. After the mulplication is done you remove 2\*7 digits from the mulplication ( either by dividing ) or by converting the numberto string and trimming the excess zeros.

## Python Implementation Notes :

### Note1 : Getting A , C , B and D

The general idea is to use integer division( //) and the modulus operator(%) , for example, if you want to split 1234 to 12 and 34, use the following pseudo-code

a = 1234 // 10^(n/2) where n is the number of digits , will give 12

b = 1234 % 10^(n/2 ) where n is the number of digits, will give 34

c = 4567 // 10^(n/2) where n is the number of digits , will give 45

d = 4567 % 10^(n/2 ) where n is the number of digits, will give 67

### Note 2: Overflows in number of digits computation

Make sure to use the int python type which can take large numbers otherwise make sure that the stored computation can be handled with the type . the zero padding can get large quickly for example the next 2\*n for number of 17 digits is 32, it already overflows the double type.

### Note 3: the extra recursive saves almost 4 seconds on three hundred thousands runs